1.1 The Different Types of Data

Data is all around us and the amount of data stored increases every single day. In today's world, decisions must be data-driven and so it is imperative that we be able to process, analyze, and understand the data we collect. Other important factors include the security and privacy of data. Businesses and governments need to answer important questions such as "Where should this data be stored?"; "How should this data be stored?"; and even, "Should this data be stored at all?". The answers to these questions for Health Canada and personal health data is very different from the answers Nintendo might come up with for the next Animal Crossing game.

We begin our study of computer science by developing definitions for different categories of data. A **data type** is a way of categorizing data. A description of a data type conveys two important pieces of information:

1. The allowed *values* for a piece of data.
2. The allowed *operations* we can perform on a piece of data.

For example, we could say that a person's age is a natural number, which would tell us that values like 25 and 100 would be expected, while an age of -2 or "David" would be nonsensical. Knowing that a person's age is a natural number also tells us what operations we could perform (e.g., "add 1 to the age"), and rules out other operations (e.g., "sort these ages alphabetically").

In this section, we'll review the common data types that we'll make great use of in this course: numeric data, boolean data, textual data, and various forms of collections of data. Many terms and definitions may be review from your past studies, but be careful—they may differ slightly from what you've learned before, and it will be important to get these definitions exactly right.

Numeric data

Here are some types of numeric data, represented as familiar sets of numbers.

- A **natural number** is a value from the set $\{0, 1, 2, \dots\}$. We use the symbol $\mathbb{N}$ to denote the set of natural numbers.[1]

[1] Note that our convention in computer science is to consider 0 a natural number!

- An **integer** is a value from the set $\{\dots, -2, -1, 0, 1, 2, \dots\}$. We use the symbol $\mathbb{Z}$ to denote the set of integers.
- A **rational number** is a value from the set $\{\frac{p}{q} \mid p, q \in \mathbb{Z} \text{ and } q \neq 0\}$—that is, the set of possible fractions. We use the symbol $\mathbb{Q}$ to denote the set of rational

numbers.

- An **irrational number** is a number with a infinite and non-repeating decimal expansion. Examples are π, e, and $\sqrt{2}$. We use the symbol $\overline{\mathbb{Q}}$ to denote the set of irrational numbers.
- A **real number** is either a rational or irrational number. We use the symbol $\mathbb{R}$ to denote the set of real numbers.

Operations on numeric data

All numeric data types support the standard arithmetic operations (addition, subtraction, multiplication, division, and exponentiation), as well as the standard comparisons for equality (using $=$) and inequality ($<$, $\leq$, $>$, $\geq$). And of course, you are familiar with many more numeric functions, like log and sin; these will come up throughout the course.

One additional arithmetic operation that may be less familiar to you is the *modulo operator*, which produces the remainder when one integer is divided by another. We'll use the percent symbol $\%$ to denote the modulo operator, writing $a \% b$ to mean "the remainder when a is divided by b". For example, $10 \% 4 = 2$ and $30 \% 3 = 0$.

Some arithmetic operations are undefined for particular numbers; for example, we can't divide by zero, and we can't take the square root of a negative number.

Boolean data

A **boolean** is a value from the set $\{\text{True}, \text{False}\}$. Think of a boolean value as an answer to a Yes/No question, e.g. "Is this person old enough to vote?", "Is this country land-locked?", and "Is this service free?".

Operations on boolean data

Booleans can be combined using *logical operators*. The three most common ones are:

- **not**: reverses the value of a boolean. "not True" is False, and "not False" is True.
- **and**: takes two boolean values and produces True when both of the values are True, and False otherwise. For example, "True and False" is False, while "True and True" is True.
- **or**: takes two boolean values and produces True when at least one of the values is True, and False otherwise. For example, "True or False" is True, while "False or False" is False.

Next week, we'll discuss these logical operators in more detail and introduce a few others.

Textual data

A **string** is an ordered sequence of characters, and is used to represent text. A character can be more than just an English letter (a, b, c, etc.): number digits, punctuation marks, spaces, glyphs from non-English alphabets, and even emojis are all considered characters, and can be part of strings. Examples include a person's name, your chat log, and the script of Shakespeare's *Romeo and Juliet*.

Writing textual data

We typically will surround strings with single-quotes to differentiate them from any surrounding text, e.g., 'David'.[2]

[2] We can also use double-quotes ("David") to surround a string, but in this course we will generally prefer single-quotes for a reason we'll discuss in Section 1.3.

A string can have zero characters; this string is called the *empty string*, and is denoted by '' or the symbol ϵ.

Operations on textual data

Here are some common operations on strings.[3]

[3] s, s_1, and s_2 are all variables representing strings.

- $|s|$: **string length/size**. Returns the the number of characters in s.

- $s_1 = s_2$: **string equality**. Returns whether s_1 and s_2 have the same characters, in the same order.

- $s + t$: **string concatenation**. Returns a new string consisting of the characters of s followed by the characters of t. For example, if s_1 represents the string 'Hello' and s_2 represents the string 'Goodbye', then $s_1 + s_2$ is the string 'HelloGoodbye'.

- $s[i]$: **string indexing**. Returns the i-th character of s, where indexing starts at 0. (So $s[0]$ returns the first character of s, $s[1]$ returns the second, etc.) For example, if s represents the string 'Hello', then $s[0]$ is 'H' and $s[4]$ is 'o'.

Set data (unordered distinct values)

A **set** is an unordered collection of zero or more distinct values, called its **elements**. Examples include: the set of all people in Toronto; the set of words of the English language; and the set of all countries on Earth.

Writing sets

We write sets using curly braces in two different ways:

1. Writing each element of the set within the braces, separated by commas. For example, $\{1, 2, 3\}$ or $\{\text{'hi'}, \text{'bye'}\}$.
2. Using *set builder notation*, in which we define the form of elements of a set using variables. We saw an example of this earlier when defining the set of rational numbers, $\{\frac{p}{q} \mid p, q \in \mathbb{Z} \text{ and } q \neq 0\}$.

A set can have zero elements; this set is called the *empty set*, and is denoted by $\{\}$ or the symbol $\emptyset$.

Operations on set data

Here are some common set operations.[4]

[4] A and B represent sets.

- $|A|$: returns the **size** of set A, i.e., the number of elements in A.
- $x \in A$: returns True when x is an element of A; $y \notin A$ returns True when y is *not* an element of A.
- $A \subseteq B$: returns True when every element of A is also in B. We say in this case that A is a **subset** of B.

  A set A is a subset of itself, and the empty set is a subset of every set: $A \subseteq A$ and $\emptyset \subseteq A$ are always True.
- $A = B$: returns True when A and B contain the exact same elements.

The following operations return sets:

- $A \cup B$, the **union** of A and B. Returns the set consisting of all elements that occur in A, in B, or in both.

  Using set builder notation: $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$.
- $A \cap B$, the **intersection** of A and B. Returns the set consisting of all elements that occur in both A and B.

  Using set builder notation: $A \cap B = \{x \mid x \in A \text{ and } x \in B\}$.
- $A \setminus B$, the **difference** of A and B. Returns the set consisting of all elements that are in A but that are not in B.

  Using set builder notation: $A \setminus B = \{x \mid x \in A \text{ and } x \notin B\}$.
- $A \times B$, the **(Cartesian) product** of A and B. Returns the set consisting of all *pairs* (a, b) where a is an element of A and b is an element of B.

  Using set builder notation: $A \times B = \{(x, y) \mid x \in A \text{ and } y \in B\}$.
- $\mathcal{P}(A)$, the **power set** of A, returns the set consisting of all subsets of A.[5] For

  [5] Food for thought: what is the relationship between $|A|$ and $|\mathcal{P}(A)|$?

  example, if $A = \{1, 2, 3\}$, then

$$\mathcal{P}(A) = \big\{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\big\}.$$

Using set builder notation: $\mathcal{P}(A) = \{S \mid S \subseteq A\}$.

List data (ordered values)

A **list** is an ordered collection of zero or more (possibly duplicated) values, called its elements. List data is used instead of a set when the elements of the collection should be in a specified order, or if it may contain duplicates. Examples include: the list of all people in Toronto, ordered by age; the list of words of the English language, ordered alphabetically, and the list of names of students at U of T (two students may have the same name!), ordered alphabetically.

Writing lists

Lists are written with square brackets enclosing zero or more values separated by commas. For example, $[1, 2, 3]$.

A list can have zero elements; this list is called the *empty list*, and is denoted by $[]$.

Operations on list data

Here are some common list operations.[6]

[6] A and B represent lists.

- $|A|$: returns the **size** of A, i.e., the number of elements in A (counting all duplicates).
- $x \in A$: same meaning as for sets.
- $A = B$: A and B have the same elements in the same order.
- $A[i]$: **list indexing**. Returns the i-th element of A, where the indexing starts at 0. So $A[0]$ returns the first element of A, $A[1]$ returns the second, etc.
- $A + B$: **list concatenation**. Returns a new list consisting of the elements of A followed by the elements of B. This is similar to set union, but duplicates are kept, and order is preserved.

  For example, $[1, 2, 3] + [2, 4, 6] = [1, 2, 3, 2, 4, 6]$.

Mapping data

Finally, a **mapping** is an unordered collection of pairs of values. Each pair consists of a *key* and an associated *value*; the keys must be unique in the mapping, but the values can be duplicated. A key cannot exist in the mapping without a corresponding value.

Mappings are used to represent associations between two collections of data. For example: a mapping from the name of a country to its GDP; a mapping from student number to name; and a mapping from food item to price.

Writing mappings

We use curly braces to represent a mapping.[7] Each key-value pair in a mapping is written

[7] This is similar to sets, because mappings are quite similar to sets. Both data types are unordered, and both have a uniqueness constraint (a set's elements are unique; a mapping's keys are unique).

using a colon, with the key on the left side of the colon and its associated value on the right. For example, here is how we could write a mapping representing the menu items of a restaurant:

$$\{\text{`fries'} : 5.99, \text{`steak'} : 25.99, \text{`soup'} : 8.99\}.$$

Operations on mappings

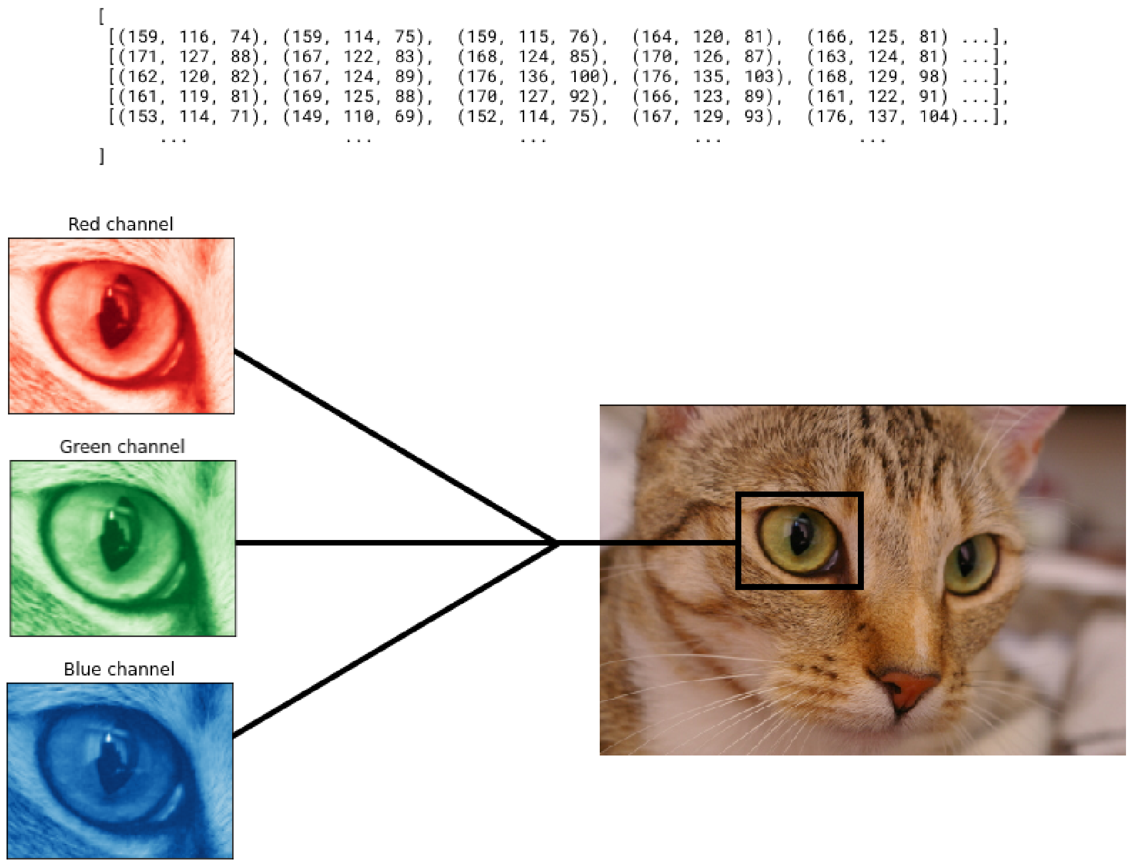
Here are some common set operations.[8]

[8] M and N represent mappings.

- $|M|$: returns the **size** of the mapping M, i.e., the number of key-value pairs in M.
- $M = N$: returns whether two mappings are equal, i.e., when they contain exactly the same key-value pairs.
- $k \in M$: returns whether k is a *key* contained in the mapping M.
- $M[k]$: when k is a key in M, this operation returns the value that corresponds to k in the mapping M.

...and more!

The data types we've studied so far are not the only kinds of data that we encounter in the real world, but they do form a basis for representing all kinds of more complex data. We'll study how to represent more complex forms of data later in this course, but here's one teaser: representing image data.



```
[
 [(159, 116, 74), (159, 114, 75),  (159, 115, 76),  (164, 120, 81),  (166, 125, 81) ...],
 [(171, 127, 88), (167, 122, 83),  (168, 124, 85),  (170, 126, 87),  (163, 124, 81) ...],
 [(162, 120, 82), (167, 124, 89),  (176, 136, 100), (176, 135, 103), (168, 129, 98) ...],
 [(161, 119, 81), (169, 125, 88),  (170, 127, 92),  (166, 123, 89),  (161, 122, 91) ...],
 [(153, 114, 71), (149, 110, 69),  (152, 114, 75),  (167, 129, 93),  (176, 137, 104)...],
      ...             ...              ...              ...              ...
]
```

Images can be represented as a list of integers. Each element in the list corresponds to a very tiny dot on your screen—a *pixel*. For each dot, three integer values are used to represent three colour channels: red, green, and blue. We can then add these channels together to get a very wide range of colours (this is called the RGB colour model). Somehow, our computers are able to take these sequences of integers and translate them into a sequence of visible lights and if these lights are arranged in a particular way, well, a cat appears!

References

1. Check out Our World in Data to see how data-driven research is being used to tackle global problems.
2. If you'd like to read more about the RGB colour model, the Wikipedia entry is a good start: https://en.wikipedia.org/wiki/RGB_color_model.